CROSS_ENTERPRISE_DOCUMENT_SHARING

XDS CODE CAMP

ONE DAY. RAW CODE.
NO EXCEPTIONS

LAKESIDE

DRILL_INSTRUCTORS

KJELD_FROBERG

BJARNE_HANSEN

LAKESIDE

CODE_COVERAGE

* **“hello, world!\n”**
  Introductory example of web service (client and server) using the DGWS profile used in many Danish healthcare applications. The example includes how to call the secure token service (STS) to obtain a validated and enriched set of credentials.
* **Administrative Tasks**
  Explanation of the necessary administrative tasks to achieve the prerequisites for development as well as access to the test systems.
* **IHE XDS and How it Works in a Danish Context**
  An overview of the general cross enterprise document sharing concept from IHE, as well as an introduction to specific Danish profiles and proprietary extensions.
* **Search and Retrieve Document**
  Examples of how to search and retrieve documents from an IHE XDS compliant service.
* **Provide and Update Document**
  Examples of how to register a new document, and how to later change it.
* **Clinical Documents**
* **Summary**
  Short summary and introduction to the compendium of tips, tricks, code examples, and links to other resources.

“hello,_world!\n”

“hello_world!\n”

TAKE_ONE: PLAIN_PYTHON

During this code camp we will be using different programming languages, but python will be predominant.

We will use the “Flask” framework to create a simple demonstration server

```
>pip install flask


from flask import Flask, request
@app.route("/", methods=["GET", "POST"])
def root():
    print("Request URL: " + request.url)
    print("Query String: " + request.query_string.decode('utf-8') if request.query_string is not None else "")
    if request.args.get("message", None) is not None:
        return request.args.get("message")
    else:
        return "Hello world!"
```

Let’s run **`simple_server.py`** and **`simple_request.py`** from **`/examples`** directory ...

“hello_world!\n”

TAKE_TWO: DGWS_INTRODUCTION (I)

* DGWS (Den Gode Web Service / The Good Web Service)
  * Currently the go-to profile for web service development in Danish healthcare IT
  * Using XMLDSIG for signing SAML Assertion using X.509 based keys/certificates
  * National Danish solution provides authoritative X.509 PKI infrastructure
* Different types of certificates: MOCES, VOCES, FOCES
* Levels: 1 = No credentials, 2 = Username/Password, 3 = VOCES/FOCES 4 = MOCES
* Standard:
  https://www.medcom.dk/standarder/webservice-standarder/den-gode-webservice
* Resources:
  https://svn.medcom.dk/svn/releases/Standarder/DGWS/
  (Documentation, Schemas, Examples, etc.)

PLEASE NOTE: DGWS is basically a profilation of how to use WS-Security with standard X.509 certificate in a Danish healthcare context.

The overall XML structure we need to generate is outlined to the right.

WS-Security Header:

```
wsse:Security
  wsu:Timestamp
  saml:Assertion (IDCard)
    saml:Issuer
    saml:Subject
    saml:Conditions
    saml:AttributeStatement (IDCardData)
      saml:Attribute
        ...
    saml:AttributeStatement (UserLog)
      saml:Attribute
        ...
    saml:AttributeStatement (SystemLog)
      saml:Attribute
        ...
```

“hello_world!\n”

TAKE_TWO: DGWS_INTRODUCTION (II)

The SAML AttributeStatement convey information about identity, user, and system involved in the communication.

IDCardData

sosi:IDCardID	Unique id of ID card.
sosi:IDCardVersion	Version of ID card (1.0)
sosi:IDCardType	Type of ID card (“user” or “system”)
sosi:AuthenticationLevel	Authentication level: 1 = No authentication 2 = Username/password 3 = VOCES/FOCES certificate 4 = MOCES
sosi:OCESCertHash	SHA1 hash of signing certificate

UserLog

medcom:UserCivilRegistrationNumber	CPR number of user.
medcom:UserGivenName	Given name of user.
medcom:UserSurName	Family name of user.
'medcom:UserEmailAddress	Email address
medcom:UserRole	Role (service dependant)
medcom:UserOccupation	Profession/occupation/title
medcom:UserAuthorizationCode	Healthcare authorisation id issued by the government.
medcom:CareProviderID	Organisational id.

SystemLog

medcom:ITSystemName	Name of IT system issuing request.
medcom:CareProviderID	Organisational id.
medcom:CareProviderName	Name of organisation issuing request.

“hello_world!\n”

TAKE_TWO: DGWS_INTRODUCTION (III)

XML-DSIG is used for signing the SAML Assertion element. The digital signature proves that the sender of the request holds the private key of the certificate used.

The signature is placed as the last child element of the SAML Assertion element (enveloped signature)

```
saml:Assertion (IDCard)
  saml:Issuer
  saml:Subject
  saml:Conditions
  saml:AttributeStatement (IDCardData)
    saml:Attribute
      ...
  ds:Signature
      ...


ds:Signature
  ds:SignedInfo
    ds:CanonicalizationMethod
    ds:SignatureMethod
    ds:Reference (#IDCard)
      ds:Transforms
        ds:Transform
          ...
      ds:DigestMethod
      ds:DigestValue
  ds:SignatureValue
  ds:KeyInfo
    ds:X509Data
      ds:X509Certificate
```

“hello_world!\n”

TAKE_TWO: DGWS_INTRODUCTION (IV)

DGWS, WS-Security, SAML, and XML-DSIG … all this and more … in Python …

* Central libraries
  * XML creation/parsing/xpath: See **lxml** module (https://lxml.de/)
  * Cryptography and certificates: See **cryptography** (https://cryptography.io/en/latest/)
  * XML Digital Signature: See **xmlsigner** module (https://pypi.org/project/signxml/)
  * Universal Unique Identifiers (UUID): See standard **uuid** module.
* Model classes
  * **pyseal.security** - classes related to WS-Security
  * **pyseal.saml** - classes related to SAML
  * **pyseal.soap** - classes related to SOAP
  * **pyseal.xml** - `to_xml(...)` function for serializing model class instances to XML

Let us walk through the code and run **`pyseal_dgws.py`** in the **`example`** directory ...

“hello_world!\n”

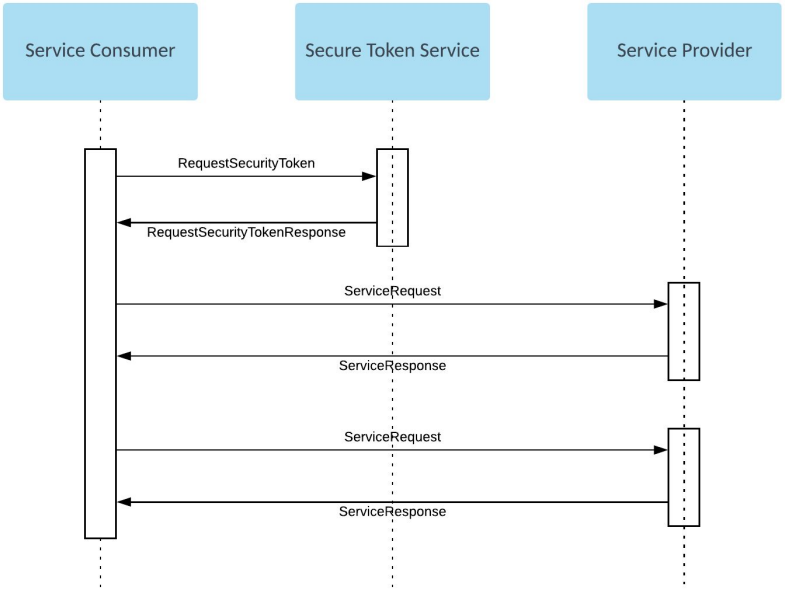
TAKE_THREE:WS_TRUST_AUTHENTICATION_FLOW (I)

If we use “plain” WS-Security each service would need to trust the individual certificates use to sign call to a service. To establish a common trust relationship we use a WS-Trust secure token service that converts our DGWS signed assertion into an assertion signed by the STS.

1. Create RequestSecurityToken.
2. Call STS to get centrally signed SAML Assertion back.
3. Call service using the centrally signed assertion.
4. …

The token issued by the STS will have a validity period in which several requests can be made without renewing the signed SAML assertion.

Where do we see the validity period? Stay tuned!



“hello_world!\n”

TAKE_THREE:WS_TRUST_AUTHENTICATION_FLOW (II)

Tasks

* We call the central STS RequestSecurityToken with a DGWS style SAML Assertion
* The STS issues a new SAML Assertion signed by the STS
* The STS provides authoritative master-data about the subject.
* The centrally signed SAML Assertion is used in subsequent calls to other services

We have the code for creating the SAML Assertion, so we just need the WS-Trust code to make the RequestSecurityToken request.

Let’s explore **`pyseal_sts.py`** in the **`/example`** directory to see how to get a centrally signed assertio

“hello_world!\n”

TAKE_THREE:WS_TRUST_AUTHENTICATION_FLOW (III)

To see how the centrally signed SAML Assertion let us call a service that provides basic healthcare information about an individual.

```
<ns2:getPersonWithHealthCareInformationIn xmlns:ns2="urn:oio:medcom:cprservice:1.0.4"
                                          xmlns="http://rep.oio.dk/cpr.dk/xml/schemas/core/2005/03/18/">
  <PersonCivilRegistrationIdentifier>0202999573</PersonCivilRegistrationIdentifier>
</ns2:getPersonWithHealthCareInformationIn>
```

This should provide us with basic master-data about a fictitious character “Anders Larsen”.

Let’s explore **`pyseal_cpr.py`** in the **`/example`** directory to see how this works.

ADMINISTRATIVE_TASKS

ADMINISTRATIVE_TASKS

A few administrative, but necessary prerequisites, must be obtained in order to get access to the National Service Platform (NSP) and the document sharing infrastructure.

* Test Service Agreement with the NSP
* Access to OCES-certificates
* Allocation and creation of master-data
* Technical whitelisting
* Production agreement for the healthcare-VPN
* Production Service Agreement with the NSP

SERVICE_AGREEMENT_WITH_THE_NSP

TEST_ENVIRONMENT

The first step is to gain access to the NSP test-environment, for that you need an approved Service Agreement with the NSP Operator.

See: Agreement with the NSP operator

Here you need to specify the services you need access to, and the purpose of the usage.

Processing time is 2-5 days

ACCESS_TO_OCES_CERTIFICATES

A prerequisite to use the services on the NSP, is that all access must be authorized using an OCES-certificate (Danish Profile of X.509 Certificates)

To acquire access to OCES-certificates, the organisation must have an agreement with the OCES operator.

See: Access to test-certificates

Processing time is 3-10 days

ALLOCATION_AND_CREATION_OF_MASTER-DATA

Some of the services at the NSP contains static test-data. That is CPR-number, Name and Address on test-citizens and allocation of ID's for danish health-personnel.

Organisations can allocate some of this test-data on https://stamdata.nspop.dk - a username and password can be provided by a service-request, see: Static and dynamic test-data

Processing time is 2-5 days

Dynamic test-data, for example CDA documents, must be provided by or coordinated among the client-system owners themselves.

TECHNICAL_WHITELISTNING

Technical whitelisting should happen automatically. When the agreement with the NSP-operator is approved.

Ensure that the certificates being used, has the same CVR number as provided in the Service Agreement with the NSP.

PRODUCTION_AGREEMENT

FOR HEALTHCARE_VPN

All services in the production environment can be accessed through the healthcare VPN (Sundhedsdatanettet or SDN).

In order to gain access to SDN you must have an agreement, see: https://www.medcom.dk/systemforvaltning/sundhedsdatanet-sdn/startpakke

When the agreement is accepted, services can be created and attached to the agreement, all this happens in the agreementsystem: https://portal.aftalesystem.dk/

When an agreement is accepted in the agreementsystem, a routing rule is created and deployed in the SDN trafficmanagement-system

SERVICE_AGREEMENT_WITH_THE_NSP

PRODUCTION_ENVIRONMENT

The last step is to gain access to the NSP production-environment, for that you also need an approved Service Agreement with the NSP Operator.

See: Production agreement with the NSP operator

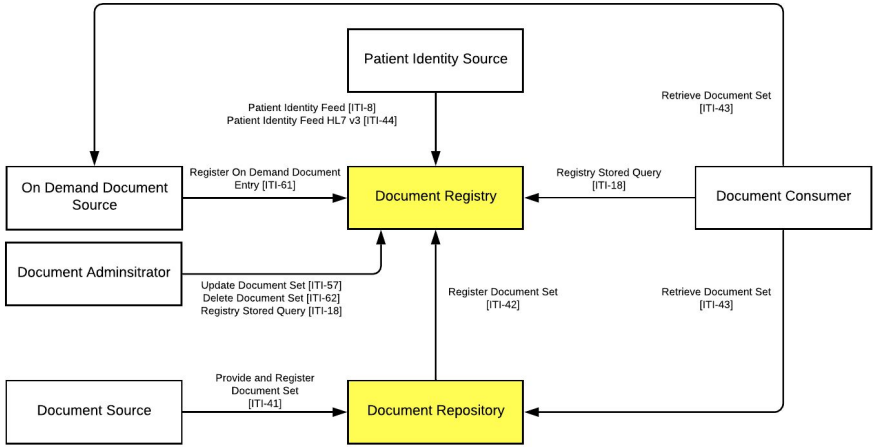
Again you need to specify the services you need access to, and the purpose of the usage.

Processing time is 3-8 days

CROSS_ENTERPRISE_DOCUMENT_SHARING

XDS

XDS_OVERVIEW



The **XDS** profile is, from a high-level perspective, just a few web services implemented by it-components.

The two central parts of XDS is the **REGISTRY** and the **REPOSITORY**.

The **REGISTRY** can be compared to old fashioned library index cards holding few meta-data informations on books in the library, as well as a reference to the bookshelf where the book itself is kept.

A **REPOSITORY** is like a bookshelf where books are stored. As in the library we can have several bookshelves storing books that are referenced from the same set of index cards.

As such we can have several **REPOSITORIES** having documents registered in the same **REGISTRY**.

XDS_SERVICES

As an application that only **SEARCH** and **RETRIEVE** documents from a registry/repository - also known as a **DOCUMENT CONSUMER** - you only need **TWO** web services!

As an application that want to **PROVIDE** documents to a repository - also known as a **DOCUMENT PROVIDER** - you need to know **ONE** additional web service.

Please observe that registering the document provided by you with the registry is actually a responsibility of the repository where you store your document.

Knowing about **TWO** more web services enables you take the role of **DOCUMENT ADMINISTRATOR** allowing you to **UPDATE** and **DELETE** documents.

Not too bad - right?

Use Case	IHE XDS Technical Name
Search for Documents	ITI-18 Registry Stored Query
Retrieve Documents	ITI-43 Retrieve Document Set
Provide (store) Documents	ITI-41Provide and Register Document Set
Update Existing Documents	ITI-57 Update Document Set
Delete Existing Documents	ITI-62 Delete Document Set

SOME_BACKGROUND

IHE XDS is based on OASIS ebXML Registry.

As such, IHE XDS is primarily a profiling of a specific usage of the enXML Registry. A few technical additions have been added, primarily in the area of repositories.

The ebXML Registry specification and the IHE XDS profile has obviously no relationship with the Danish profile for web services (DGWS).

However, the result is that a significant part of the XML structures comes from the ebRS XML schemas.

Let us take a look at the XML schemas for ebRS and IHE XDS ...

REGISTRY_STORED_QUERY

CALLING_ITI-18 (I)

We build a so-called `AdhocQueryRequest` ...

The central part is the Slot XML element which holds the query parameters. Let us see an example ...

Let us consult **`IHE_ITI_TF_Vol2a.pdf`** from the **`/doc`** directory for a full list of query parameters … they are in section “3.18.4.1.2.3.7.1 FindDocuments”
… please observe that they have a “multiplicity” column to the right … we’ll see that in code in a second ...

Where do these data come from? Let us consult the Danish IHE Meta-Model Standard found in **`Metadata-v096.pdf`** in the **`/doc`** directory.
Specifically, table 4 on page 15.

```
AdhocQueryRequest
  ResponseOption
  AdhocQuery
    Slot
    ...
```

Example Slot (service start time):

```
<rim:Slot name="$XDSDocumentEntryServiceStartTimeFrom">
  <rim:ValueList>
    <rim:Value>20171231000000</rim:Value>
  </rim:ValueList>
</rim:Slot>
```

REGISTRY_STORED_QUERY

CALLING_ITI-18 (II)

Let's take a look at the weird code with names like *classCode*, *typeCode*, *practiceSettingCode*, *healthcareFaclityTypeCode* … in the Meta-Data model ...

We can find value sets for these in:
`DK-IHE_Metadata-Common_Code_systems-Value_sets.xlsx`

Also, let us have a short look at some data types causing our request to look a bit bizarre … in ...
`IHE_ITI_TF_Rev8-0_Vol3_FT_2011-08-19.pdf`, look at table 4.1-3 Data Types

'nuff said … show the code! … it is in
`iti18_appointments.py` in **`/src/examples/iti_18`**

classCode: high-level classification
typeCode: type of document
practiceSettingCode: clinical speciality for document
healthcareFacilityTypeCode: healthcare organization
...

What's up?

```
<rim:Slot name="$XDSDocumentEntryTypeCode">
  <rim:ValueList>
    <rim:Value>('39289-4^^2.16.840.1.113883.6.1')</rim:Value>
  </rim:ValueList>
</rim:Slot>
```

REGISTRY_STORED_QUERY

CALLING_ITI-18 (III)

The response is an `AdhocQueryResponse` XML structure with a number of `ExtrinsicObject` elements, each containing information about a document matching our query …

To retrieve the actual document - using ITI-43 Retrieve Document Set - we need to extract 3 values ...

- Home Community Id
  `//ExtrinsicObject/@home`
- Repository Unique Id
  `//ExtrinsicObject/Slot[@name='repositoryUniqueId']/ValueList/Value`
- Document Unique Id
  `//ExtrinsicObject/ExternalIdentifier[@identificationScheme='urn:uuid:2e82c1f6-a085-4c72-9da3-8640a32e42ab]/@value`

```
AdhocQueryResponse
  RegistryObjectList
    ExtrinsicObject
    ...
```

RETRIEVE_DOCUMENT_SET

CALLING ITI-43 (I)

We build a so-called `RetrieveDocumentSetRequest` ...

We can retrieve more documents in one request by specifying a `DocumentRequest` element for each of the documents we want to retrieve.

The `DocumentRequest` holds the 3 central values needed to retrieve a document.

Let us check out the code in **`iti43_appointments.py`**

```
RetrieveDocumentSetRequest
  DocumentRequest
    HomeCommunityId
    RepositoryUniqueId
    DocumentUniqueId
  ...
```

RETRIEVE_DOCUMENT_SET

CALLING ITI-43 (II)

The response is MIME multipart/related

Root part holds the SOAP response and following parts each hold a document.

We need to parse the MIME response and inspect the `RegistryResponse` element to check if request was successful.

Each DocumentResponse holds a reference to the content id of the MIME part holding document content, for example:

```
<Include
href="cid:8110ae30-37ff-4306-9c98-0ec153181a9f-89545@urn%3Aihe%3Aiti%3Axds-b%3A2007"/>
```

```
Connection: keep-alive
X-Powered-By: Undertow/1
Server: WildFly/8
Transfer-Encoding: chunked
Content-Type: multipart/related; type="application/xop+xml";
boundary="uuid:6ddd018f-51a0-4233-a170-2fe87fbefd3f";
start="<root.message@cxf.apache.org>"; start-info="text/xml"
Date: Sun, 16 Jun 2019 14:35:08 GMT

--uuid:6ddd018f-51a0-4233-a170-2fe87fbefd3f
Content-Type: application/xop+xml; charset=UTF-8; type="text/xml"
Content-Transfer-Encoding: binary
Content-ID: <root.message@cxf.apache.org>

<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">...
--uuid:6ddd018f-51a0-4233-a170-2fe87fbefd3f
Content-Type: application/octet-stream
Content-Transfer-Encoding: binary
Content-ID: <8110ae30-37ff-4306-9c98-0ec153181a9f-89545@urn:ihe:iti:xds-b:2007>

<ClinicalDocument xmlns="urn:hl7-org:v3">...
--uuid:6ddd018f-51a0-4233-a170-2fe87fbefd3f--


SOAP Body of response:
RetrieveDocumentSetResponse
  RegistryResponse
  DocumentResponse
    HomeCommunityId
    RepositoryUniqueId
    DocumentUniqueId
    mimeType
    Document
      Include
  ...
```

PROVIDE_AND_REGISTER_DOCUMENT_SET

CALLING ITI-41 (I)

ProvideAndRegisterDocumentSet is a request sent to the **REPOSITORY** in order to store a new document. The repository will then issue a RegisterDocumentSet (ITI-42) against the corresponding **REGISTRY**.

We remember the `ExtrinsicObject`, `Slot`, `Classification`, and `ExternalIdentifier` from our ITI-18 responses … remember the Danish profile for XDS metadata? So, this is where they come from …

The `RegistryPackage` helps telling the registry information about the submission set.

The `Association` binds one or more documents into the submission set.

Finally, one or more `Document` elements specifies a reference to the actual document content.

```
ProvideAndRegisterDocumentSetRequest
  SubmitObjectRequest
    RegistryObjectList
      ExtrinsicObject
        Slot
        ...
        Classification
        ...
        ExternalIdentifier
        ...
      RegistryPackage
        Slot
        ...
        Classification
        ...
        ExternalIdentifier
        ...
      Classification (here we tell it is a submission set)
      Association
        Slot
        ...
  Document
  ...
```

PROVIDE_AND_REGISTER_DOCUMENT_SET

CALLING ITI-41 (II)

The `ProvideAndRegisterDocumentSet` request **AND** the actual document content is sent as a multipart/related request to the repository.

Let us have a look at he code ...

```
POST /drs/proxy HTTP/1.1
Host: test2-cnsp.ekstern-test.nspop.dk:8080
User-Agent: python-requests/2.21.0
Accept-Encoding: gzip, deflate
Accept: */*
Content-Type: multipart/related; type="application/xop+xml";
boundary="8594a829-78f8-4e1c-8672-3f1ccc2a6e26"; start="<root@xds.lakeside.dk>";
start-info="text/xml"
Content-Length: 14968
Connection: keep-alive

--8594a829-78f8-4e1c-8672-3f1ccc2a6e26
Content-Type: text/xop+xml; charset="UTF-8"; type="text/xml"
Content-Transfer-Encoding: binary
Content-ID: <root@xds.lakeside.dk>

<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">...
--8594a829-78f8-4e1c-8672-3f1ccc2a6e26
Content-Type: text/octet-stream
Content-Transfer-Encoding: binary
Content-ID: <6145a062-ad73-4d48-9901-d27be33ec9c6@urn:ihe:iti:xds-b:2007>

Hello world!
--8594a829-78f8-4e1c-8672-3f1ccc2a6e26--
```

CLINICAL_DOCUMENTS

CLINICAL_DOCUMENT_ARCHITECTURE_2.0

While any kind of document can be stored in a repository and registered in a registry, many clinical documents are stored in a XML format HL7 CDA version 2.0.

The document format is based a set of basic building blocks that are used to compose a document standard. This causes CDA documents to to be quite verbose and complicated to read.

In Denmark, MEDCOM oversees the creation of Danish standards or profiles of HL7 CDA documents.

HL7 CDA 2.0
https://www.hl7.org/implement/standards/product_brief.cfm?product_id=7

MEDCOM
https://svn.medcom.dk/svn/releases/Standarder/HL7/

CLINICAL_DOCUMENT_ARCHITECTURE_2.0

DOCUMENT_STRUCTURE

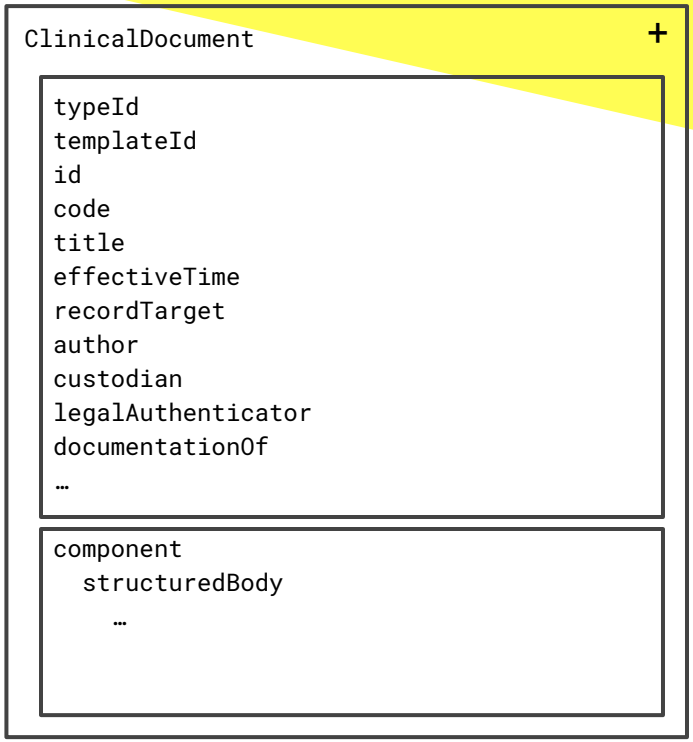
A ClinicalDocument has a header section containing basic meta-data about the document.

The meta-data in the CDA header is standardized and described in DK-CDA-Header.pdf (see MEDCOM svn).

The meta-data in the CDA header bears close resemblance with the meta-data used when registering documents in a IHE XDS Registry.

See Metadata-v096.pdf for sources of registry meta-data in CDA documents.

Let us review a PHMR document in order to see the relatively complex structure of a CDA document.



CLINICAL_DOCUMENT_ARCHITECTURE_2.0

CDA_BUILDERS

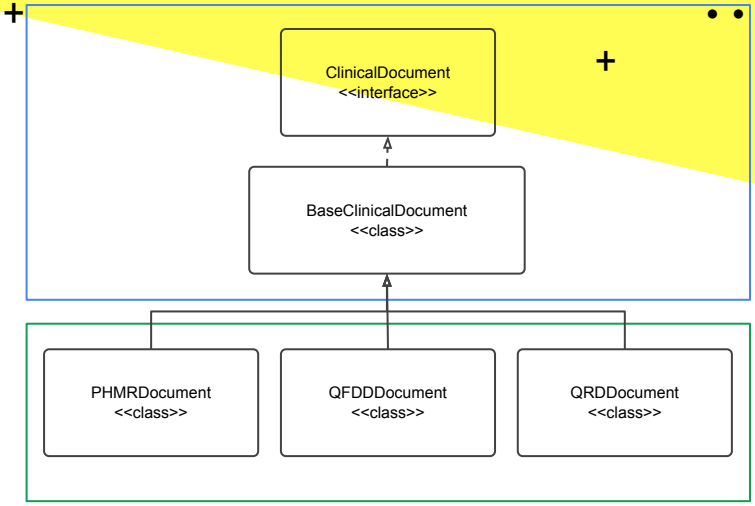
To make it easier to create the standard CDA documents used in the Danish healthcare sector a set of CDA builder components have been developed.

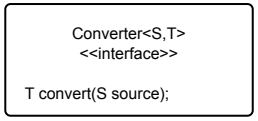
The “data model” is created to be more user-friendly than building the raw structure of a CDA document.

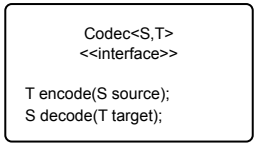
The data model is based on a inheritance hierarchy with each specific document type inheriting the common header structure from a common base class.

To serialize/deserialize an object model to/from XML we use builder components (Converters and Codecs).

See http://4s-online.dk/wiki/doku.php?id=4scdabuilder:overview
Code https://bitbucket.org/4s/4s-cda-builder/src/master/







SUMMARY

IN_SUMMARY

Summary of topics covered

- DGWS - A profile SAML profile
- STS - The WS-Trust Secure Token Service
- The Administration
- ITI-18, ITI-43, and ITI-41
- Clinical Documents

Ressource slides following this one …

Thank you very much for attending! Questions anyone?

RESOURCES

- /doc
- standards
- service_endpoints
- useful_links

freepik

Graphics in this presentation downloaded from freepik:
Designed by Pressfoto / Freepik
Designed by kues1 / Freepik
Medical photo created by freepik - www.freepik.com
Business photo created by rawpixel.com - www.freepik.com

/doc

DOWNLOADED_DOCUMENTATION

MEDCOM (http://svn.medcom.dk)
Den Gode Webservice_1.0.pdf
BILAG - Den Gode Webservice_1.0.1.pdf
Metaddata-v096.pdf
Kom-godt-igang-med-dokumentdeling 1.4 (interactive).pdf

IHE XDS
IHE_ITI_TF_Vol1.pdf
IHE_ITI_TF_Vol2a.pdf
IHE_ITI_TF_Vol2b.pdf
IHE_ITI_TF_Vol2x.pdf
IHE_ITI_TF_Rev8-0_Vol3_FT_2011-08-19.pdf
IHE_ITI_Suppl_On_Demand_Documents_Rev1.3_TI_2013-10-25.pdf

STANDARDS

IN_ORDER_OF_APPERANCE

SOAP Version 1.2
http://www.w3.org/TR/soap12-part0/ * http://www.w3.org/TR/soap12-part1/ * https://www.w3.org/TR/soap12-part2/

WS-Security 1.1
https://www.oasis-open.org/committees/download.php/16790/wss-v1.1-spec-os-SOAPMessageSecurity.pdf

XML Signature Syntax and Processing Version 2.0
https://www.w3.org/TR/xmldsig-core2/

WS-Trust 1.3
http://docs.oasis-open.org/ws-sx/ws-trust/v1.3/ws-trust.html

STS User Guide (Danish)
https://www.nspop.dk/display/public/web/STS+-+Guide+til+anvendere#STS-Guidetilanvendere-Snitflade

Introduction to the NSP platform (Danish)
https://www.nspop.dk/display/public/web/Introduktion+til+NSP-platformen

Cross Enterprise Document Sharing Wiki
https://wiki.ihe.net/index.php/Cross-Enterprise_Document_Sharing

SERVICE_ENDPOINTS

IN_ORDER_OF_APPERANCE

http://test2.ekstern-test.nspop.dk:8080/stamdata-cpr-ws/service/DetGodeCPROpslag-1.0.4

USEFUL_LINKS

Document Sharing Service (DDS) Documentation (Danish)
https://svn.nspop.dk/public/components/dds/latest/doc/

Examples of ProvideAndRegisterDocumentSet as well as a summary of defined UUIDS
http://ihewiki.wustl.edu/wiki/index.php/Notes_on_XDS_Profile